Argonne National Laboratory Technical Report ANL/MCS-P1281-0805, August 2005

Statistical Data Reduction for Efficient Application Performance Monitoring

Lingyun Yang[1] Jennifer M. Schopf[2] Ian Foster[1,2]

[1]*Department of Computer Science, University of Chicago, Chicago, IL 60637*
[2]*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439*
lyang@cs.uchicago.edu [jms, foster]@mcs.anl.gov

Abstract

There is a growing need for systems that can monitor and analyze application performance data automatically in order to deliver reliable and sustained performance to applications. However, the continuously growing complexity of high-performance computer systems and applications makes the process of performance monitoring and analysis difficult. We introduce here a statistical data reduction method that can be used to guide the selection of system metrics that are both necessary and sufficient to describe observed application behavior, thus reducing the instrumentation perturbation and data volume managed by a monitoring system. In order to evaluate our strategy, we applied it to two Grid applications. A comparative study shows that our strategy produces better results than other techniques. It can reduce the number of system metrics that need to be managed by about 80% percent, while still capturing enough information for prediction of application performance.

1 Introduction

Recent experience in deploying Grid middleware demonstrates the challenges one faces in delivering robust services in distributed and shared environments [6,13]. Applications often must deliver reliable performance despite the use of distributed and shared resources. In order to deliver dependable and sustained performance to applications given current state-of-the-art technology, there is a need for systems that can detect anomalies or bottlenecks automatically, and then adapt their behavior dynamically to address such problems in an online manner.

The first step towards this kind of fault tolerant and adaptive computing is to monitor the performance of system components such that we can diagnose the reason if an anomaly happens. However, as high-performance computer systems and applications continue to increase in complexity, performance monitoring and analysis grows more difficult. Runtime performance problems seen by applications result from interactions among both physical components (e.g., CPUs) and logical components (e.g., I/O buffers). The performance implications of each isolated component may be relatively well understood, but it is often far from clear how this web of time-varying relationships determines overall application performance.

To understand relationships among performance components and to address performance problems, developers must use instrumentation systems that capture information on a large number of these time-varying system metrics. Unfortunately, this instrumentation can influence the performance of target systems and produce tremendous volumes of data [11]. Among these system metrics, some can be predicted from others and are thus redundant. Other metrics are unrelated to application performance. To help combat these consequences of performance instrumentation, we need mechanisms to provide support for selecting only necessary metrics and measurement points. Such mechanisms will also simplify data analysis.

In this paper, we introduce how to use the a statistical data reduction strategy based on stepwise regression to aid in the selection of system metrics that are necessary and sufficient to capture the relevant information of observed application behavior. One goal of this strategy is to reduce perturbation and data volume while retaining interesting characteristics of performance data. This statistical technique, when used for performance analysis, attempts to reduce the number of system metrics that a monitoring system must manage, which, in turn, can dramatically reduce data volume and instrumentation perturbation.

We proceed in two steps. First, we identify and eliminate redundant system metrics. Second, we select the system metrics that are related to application performance, and eliminate (unnecessary) metrics. We evaluate the effectiveness of our data reduction strategies for two parallel applications, Cactus and Sweep3D, running on shared resources. Our experimental results show that our strategy selects system metrics that are both necessary and sufficient to predict application behavior, and also greatly reduces the number of system metrics collected.

The rest of this paper is organized as follows. Section 2 describes the problem. Section 3 introduces our data reduction strategy. Section 4 presents our experimental results. Section 5 introduces related work. In Section 6, we summarize and briefly discuss future work.

2 Problem Statement

Previous studies [5,10] show that variability in resource characteristics can have a major influence on application performance. Even small variations in resource capability can dramatically change observed application performance. To formalize this notion and provide a basis for analysis, we consider a distributed system with *p* resources, each characterized by a set of system metrics. For example, a resource *CPU* might have three system metrics defined: percentage of CPU utilization at the user level, percentage of CPU utilization at the system level, and percentage of time that the CPU was idle.

The characteristics of the system including *p* resources can be described by the set of all system metrics:

$$M = \begin{pmatrix} m^1{}_1, m^1{}_2, \ldots m^1{}_{n1}; \\ m^2{}_1, m^2{}_2, \ldots m^2{}_{n2}; \\ \ldots \\ m^p{}_1, m^p{}_2, \ldots m^p{}_{np}; \end{pmatrix}$$

where

ni is the number of metrics for the *i*th resource and,

$m^i{}_j$ is the *j*th metric for the *i*th resource.

We also introduce the notion of a *performance metric*, a quantitative description of some aspect of application performance. For example, one useful performance metric for an application that calculates the product of two matrices might be the number of multiplications finished during a unit time. Another might be elapsed time during program execution.

Depending on the system metrics available on a particular system and the performance metrics of interest to the user, we may find that there is a correlation between some function of some subset of the system metrics and a particular performance metric. If such a function and subset exist and can be identified, then we can use those particular system metrics as a predictor for the performance metric.

Given this introductory material, we can now state our problem as follows: given a system, a set of system metrics, and an application described by a performance metric, identify a minimal set of system metrics that can be used to predict the performance metric with a desirable level of accuracy.

In solving this problem, we can exploit the fact that some system metrics may capture the same or similar information. For example, the value of the metric *used memory* is equal to the total memory size minus the value of the metric *unused memory*, and vice versa, while the two system metrics *number of messages sent* from one socket and the *number of bytes sent* from the same socket will capture similar information if messages have similar sizes.

Definition 1: Two system metrics m_1, m_2 are *dependent* on each other if and only if the value of the two system metrics are correlated with each other at a level greater than a specified threshold. Otherwise, m_1 and m_2 are *independent*.

For example, the value of *used memory* and the value of *unused mem*ory of the same machine are dependent on each other. Only one is necessary; the other is *redundant* and can be eliminated from the set of potential predictors without losing useful information.

Thus, to judge if two variables are dependent, we need both (a) a method of quantifying the degree of correlation between two variables and (b) a value for the threshold correlation value. We discuss our method of calculating the degree of correlation between two system metrics and the selection of the threshold parameter in Section 3.1 and Section 4, respectively.

We also want to take into account the fact that not all system metrics will be related with a particular performance metric. For example, in the case of a program that calculates the sum of several integers, CPU utilization is likely to be strongly related with execution speed, while the number of opened files is not. Such unrelated system metrics can also be eliminated from the set of potential predictors without losing useful information.

To describe the relationship between system metrics and performance metric, we give the following definition:

Definition 2: A performance metric y is *predictable* by system metrics $m_1,m_2,\ldots m_n$ if and only if the value of y can be predicted by a model of variable $m_1,m_2\ldots m_n$, expressed by $y=F(x_1,x_2,\ldots x_n)$. We then call y the *response variable* and each system metric m_i (i=1..n) is called a *predictor* of y.

Given the above definitions, our problem can be formalized as follow: given an application characterized by a performance metric y and a system characterized by a set of metrics M, our goal is to find a subset of system metrics $S= (x_1,x_2,\ldots.x_n)$, $S\subseteq M$, such that (a) every pair of system metrics in S, x_i and x_j, i=1..n, j=1..n , $i\neq j$, are independent and (b) every system metric x_i, i=1..n, is a predictor of the performance metric y of the application running on this system, using a given model. The goal of criterion (a) is to remove redundant system metrics. The goal of criterion (b) is to find all metrics that predict application performance and remove those that do not.

3 Statistical Data Reduction

Recall that the general goal of our strategy is to reduce the number of system metrics to be monitored while still capturing enough information to predict a specified performance metric with a desired level of accuracy. We proceed in two steps: (1) eliminate dependent system metrics (Section 3.1) and (2) identify and further eliminate irrelevant system metrics (Section 3.2). The result of our strategy is a subset of system metrics that are both necessary for predicting the performance metric and independent of each other. In Section 3.3 we discuss the criteria used to evaluate the data reduction strategy.

3.1 Redundant System Metrics Reduction

Dependent system metrics, as defined by Definition 1, are system metrics that are strongly correlated each other. A pair of dependent metrics convey essentially the same information, thus only one is necessary; the other is redundant and can be eliminated.

We use the *Pearson Product-Moment Correlation Coefficient* (r), or correlation coefficient for short, to obtain a quantitative estimation of the degree of correlation between two system metrics. The correlation coefficient provides a measure of the degree of linear relationship between two variables [18]. A high correlation coefficient value indicates that the two variables can be calculated from each other by some linear model. The correlation coefficient may take any value between plus and minus one. The sign of the correlation coefficient (+ , -) defines the direction of the relationship, either positive or negative. The absolute value of the correlation coefficient represents the strength of the relationship.

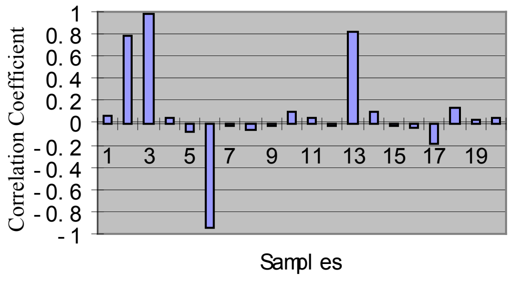
To identify system metrics that are dependent on each other, we first construct a correlation matrix by computing the correlation coefficient between every pair of system metrics. Then, we apply a clustering algorithm to this matrix to identify clusters of system metrics such that every metric in a cluster has a correlation coefficient with absolute value above a threshold value to at least one other metric in the cluster. We conclude that the system metrics in one cluster capture essentially the same information and eliminate all but one of those metrics.

Two questions then arise: what data do we use to construct this correlation matrix, and how much data do we need to determine whether an observed correlation is significant?

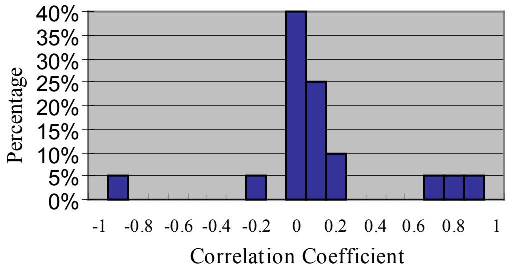
The answer to the first question is that we use data measured during some "typical" execution of our application and system. Here we see a potential weakness of our approach: while we capture effectively, as we show below, correlations between system and performance metrics that occur during "normal execution," a correlation that occurs only during some "unusual" circumstance may not be detected if our sample data includes few or no occurrences of that circumstance. We recognize this weakness, but note that it does not prevent our technique being useful in many cases.

The answer to the second question is that we use statistical tests to determine whether an observed correlation is significant, with the goal of avoiding eliminating uncorrelated metrics only by chance. To show why this is important, Figure 1 (a) shows 20 sample correlation coefficients between *the number of transfers issued per second* and *the number of memory pages cached per second* for data collected during the execution of one of our test applications, Cactus [2,3], a compute-intensive numerical simulation. We ran Cactus 20 times on six machines and collected 20 sets of sample data on one of the six machines. We then computed one sample correlation coefficient on each set of sample data. Figure 1(b) shows the histogram of the 20 sample coefficient coefficients.

Because these two system metrics describe the performance of two independent components in the system, we would not expect them to be strongly correlated. However, as shown in Figure 1, while in most cases (16 cases or 80% of the time) the absolute value of the sample correlation coefficients between these two metrics is small (<0.2), in a few cases (4 cases or 20% of the time) the absolute value is high (as high as 0.98 in our example). Thus, a correlation coefficient obtained using a single sample could cause a false positive error and group these two independent system metrics into one cluster. The consequence of this error is that we would delete useful information.



(a)Value of sample 20 correlation coefficients.



(b) Histogram of 20 sample correlation coefficients

Figure 1: Sample correlation coefficient between *the number of transfers issued per second* and the *number of memory pages cached per second* for 20 runs of cactus application.

To reduce the chance of such false positive errors, we use a one-tailed Z-test to determine whether an observed correlation is statistically significant. Z-test is a statistical method used to test the viability of hypothesis in the light of sample data with specific confidence. More specifically, in our strategy, the hypothesis to be tested is: the actual correlation coefficient is less than or equal to the threshold value. Given a set of sample data, Z-test tests the possibility of observing the sample correlation coefficient if the actual correlation coefficient between the two metrics is less than or equal to the threshold value. If the possibility is small (<5% in our work), we can reject the hypothesis with more than 95% confidence and say that the real correlation coefficient is statistically larger than the threshold value. We then group the two system metrics involved into one cluster.

Thus, given a set of samples, we proceed as follows. We perform the Z-test for correlation coefficient between every pair of system metrics, and group two metrics into one cluster only when the absolute value of their correlation coefficient is larger than the threshold value with 95% confidence. The result of this computation is a set of system metric clusters. System metrics in each cluster are strongly correlated, so only one metric from the cluster can be used as the representative of the cluster while the others are deleted as redundant. Currently, we pick as the representative the system metric with the highest correlation coefficient value with the application performance metric.

We also find it useful to identify and eliminate system metrics that have no variation in the sample data, which means that no correlation coefficient involving these metrics can be calculated. We group these metrics into a special cluster called "zero variation," in which all metrics have a variation of zero. An initial investigation found that these system metrics can be roughly classified into two categories. The first category consists of the system metrics that will not change due to the configuration of the machine. An example is system metrics describing spare network interfaces. Some machines may have multiple network interfaces, but only one of them is configured to connect to the network. Thus, the statistical information of other network interfaces will remain zero all the time. The second category consists of system metrics whose value could in principle change, but does not do so during the period of data collection. Examples of this kind of system metrics are *number of inactive pages in memory* and *number of multicast packets received per second*. Although these system metrics could be predictors for the performance metric, the lack of variation in the sample data means that we are not able to identify them as such. We consider all metrics with zero variations to be redundant metrics and immediately eliminate them, since (as far as we can determine based on our sample data) they do not carry any useful information for predicting the performance metric.

We also need to decide the value of the threshold used to judge whether the correlation between two system metrics is strong enough to put them into one cluster. The value of this parameter has a strong influence on the data reduction result. If set too low, some independent system metrics will be grouped into one cluster and deleted, and thus we lose some useful information at the early stage of the data reduction. If the threshold value is set too high, some dependent system metrics will not quality to group into one cluster, and thus some redundant system metrics will remain in the final result. We discuss our selection of this value and its influence on data reduction in Section 4.

3.2 Statistical Variable Selection

After deleting all redundant system metrics, we have a subset of system metrics that we have decided to treat as independent of each other. However some of these system metrics may not relate to our chosen performance metric. Thus, in the second step of our strategy, we identify the subset of all predictors that are necessary to predict the performance metric, further reducing system metrics that either are unrelated to the performance metric, or that, given other metrics, are not useful for predicting the performance metric. This form of data reduction is also known as *variable selection*.

Two basic methods for variable selection have evolved. The first method uses a criterion statistic computed for all possible subsets of predictors. This method is able to find the best solution but is inefficient. The second method, generally called stepwise regression, provides a systematic technique for choosing a path through the possible subsets, looking first at a subset of one size, and then looking only at subsets obtained from preceding ones by deleting one potential predictor. This limiting of the number of subsets of each size that must be considered makes the second method more efficient than the first.

We focus here on the second method. Specifically, we use the Backward Elimination (BE) stepwise regression method [20] to select our predictors. This method is a well-known variable selection technique commonly used in statistics to select a good set of predictors among many potential predictors for a response variable. We use it because of its simplicity and efficiency.

To apply BE in our data reduction problem, we treat every system metric left after the first step as a potential predictor and the application performance metric as the response variable to be predicted. We start with a model that includes all potential predictors, and at each step, delete one metric that either is irrelevant or that, given other metrics, is not useful to the model, until all metrics left are statistically significant. More specifically, the algorithm can be described as follows:

1) Build a full model by regressing the response variable on all possible predictors using a linear model:

$$Y=\beta_0+\beta_1x_1+\beta_2x_2+\ldots\beta_nx_n \qquad (1)$$

where

Y is the response variable, or performance metric in our work;

X_i (i=1…n) are all potential predictors, i.e., the independent system metrics considered in our work.

2) Pick the least significant predictor in this model by calculating the F value of every predictor in current model. The F value of a predictor captures its contribution to the model. A smaller value indicates a smaller contribution thus a less significant predictor. The F value of a predictor x is defined as the result of an F test, which assesses the statistical significance of two different models: one with all predictors considered, called the full model; the other with all the other metrics except the predictor x, called reduced model. The F value indicates how different the two models are: a small F value means there is little difference between the two models, and thus x does not make a large contribution. So we can remove it without reducing the prediction power of the model.
3) If the smallest F value is less than some predefined significant value (in our case, 2), remove the corresponding predictor from the model. Go to step 4; if not, the algorithm stops. The remaining predictors are considered to be significantly related to the response variable and necessary when predicting the response variable. We set predefined significant F value to 2 as suggested by [20].
4) Re-regress the response variable on all left potential predictors using a linear model. Go to step 2.

We note that while the BE regression method is usually employed with a linear model as defined in function 1, it need not be limited to a linear function. For example, we can add a quadratic item for every potential predictor in the model. For each system metric *x*, we not only consider *x* itself, but also treat x^2 as a potential predictor to predict the performance of application. The new regression model is:

$$Y=\beta_0+\beta_1x_1+\beta_2x_2+\ldots\beta_nx_n+\beta_{n+1}x_1^2+\beta_{n+2}x_2^2+\ldots\beta_{2n}x_n^2 \qquad (2)$$

Using this model, the BE method will select those system metrics that are either linearly *or* quadratically related to the performance metric. Quadratic terms turn out to be important in the sweep3D application that we consider in Section 4.4.

3.3 Evaluating Data Reduction Strategies

Recall that the general goal of our strategy is to reduce the number of system metrics to be monitored while still capture enough important information. We use two criteria to evaluate this data reduction strategy.

The *reduction degree* criterion is the total percentage of system metrics eliminated. This criterion measures how many system metrics are reduced by the strategy, so the larger the better. This criterion is used to ensure that we do not leave many redundant or unnecessary metrics in the results.

The *coefficient of determination* [20] criterion, denoted as R^2, uses a statistical measurement that indicates the fraction of the total variability in the response variable (performance of application in our case) that can be explained by the predictors (the system metrics in our work) in a given model. In another word, R^2 measures whether the predictors used in this model are sufficient to predict the response variable. R^2 is a scale-free number, ranging from 0 to 1, the larger the better. A small value of R^2 may be an indication that we lost useful information.

4 Experimental Evaluation

To evaluate the validity of our data reduction strategy, we run a series of comparative experiments involving two parallel programs running in a shared local area network environment.

4.1 Test Applications and Data Collection

We test our strategy on data collected on two applications: Cactus and sweep3D. Cactus [2,3] is a numerical simulation of a 3D scalar field produced by two orbiting astrophysical sources. The solution is found by finite differencing a hyperbolic partial differential equation for the scalar field. The application can run on multiple processors, with the 3D scalar field decomposed over processors and an overlap region placed on each processor. At each iteration, each processor updates its local grid point and then uses MPI message passing to synchronize the boundary values. This application's performance metric is defined as the elapsed time per iteration.

Sweep3D [8] is a solver for the 3-D time-independent, particle transport equation on an orthogonal mesh. The solver computes along wavefronts in mesh in eight diagonal directions through the cube. This application is also run on multiple processors using domain decomposition and MPI message passing and the performance metric is, again, elapsed time per iteration.

We run each application on six Linux machines and collect system metrics and the application performance metric at a frequency of 0.033 HZ (i.e., once every 30 seconds). Each data point includes one performance metric and a total of roughly 600 system metrics. All machines are shared with other users during the data collection. The performance of applications varied greatly due to resource contention during the period of data collection. During the roughly 24-hour period over which data was collected, the Cactus performance metric varied between 0.6 and 49.0 seconds (sec) per iteration, with an average of 9.2 sec and a standard deviation of 12.3 sec. The Sweep3d performance metric of varied between 1.1 and 134 sec per iteration, with an average of 4.0 sec and a standard deviation of 13 sec.

We collect system metrics on each machine using three utilities, namely (1) the *sar* command of the SYSSTAT tool set, (2) network weather service (NWS) sensors, and (3) the Unix command *ping*.

SYSSTAT [1] utilities are a collection of performance monitoring tools for Linux. The *sar* command displays the contents of selected cumulative activity counters in the operating system, including statistical information about various physical devices (e.g., network, CPU, I/O, memory, block device), logical resources (e.g., Inode, queue, processes) and system activities (e.g., swapping, paging, interrupt and switching). It can measure hundreds of system metrics.

NWS [21,22] is a distributed system that periodically monitors and dynamically forecasts the performance of various network and computational resources. It provides measurements of seven system metrics concerning CPU, memory, disk, and network behavior.

Ping is a commonly used UNIX command that uses timed ICMP ECHO_REQUEST packets to measure the path latency to the target machine. We measure the ping values between every pair of the six target machines.

4.2 Experimental Methodology

We divided the data collected into two disjoint sets: the training data and the verification data. The first step of our experiment involves using our data reduction strategy on the training data to select a subset of system metrics that are both necessary and sufficient to capture the application behavior. Recall that our data reduction strategy needs a threshold parameter (Section 3.1) to determine whether two system metrics are qualified to be grouped into one cluster. In this step, we also evaluate influence of the threshold parameter on our data reduction strategy by exhaustively searching the space of feasible selections. Our results are given in the Section 4.3 and Section 4.4.

In the second step of our experiment, we evaluate the efficiency of our strategy using the verification data. We compare the result of our statistical data reduction strategy (**SDR**) to two other strategies. The first method, **RAND**, randomly picks a subset of system metrics equal in number to those selected by our strategy. The second method, **MAIN**, uses a subset of 75 system metrics that are commonly used to model the performance of applications [4,17]. More specifically, the MAIN metrics include: (1) network measurements, including *bandwidth*, *latency* and *the time required to establish a TCP connection* between every pair of the six machines; (2) CPU measurements, including the *fraction of CPU available to a process that is already running* and *to a newly started process* on every machine, and *the system load average for the last minute*; (3) *the amount of space unused in memory* on every machine; and (4) *the amount of space unused on the disk* of every machine.

In the second step, we use the coefficient of determination (R^2) to determine whether the system metrics selected on the training data are sufficient to capture the application behavior. As noted above, R^2 indicates the fraction of the total variability in the application performance that can be explained by the system metrics considered. A small value may be an indication that the system metrics selected are not sufficient, thus the strategy is less effective.

4.3 Cactus Results

We ran Cactus on six shared Linux machines at the University of California, San Diego, and collected data over roughly a one day period, partitioned into 12 roughly equal-sized chunks, each containing about 2 hour's data. Every data point comprises the values of 628 system metrics and one application performance metric. A detailed description of the system metrics used is available online [23].

We used the first chunk of data (with 240 data points) as the training data to select the necessary and sufficient system metrics, while varying the threshold value to evaluate its influence on the data reduction result. The threshold value is evaluated at intervals of 0.05 between 0 and 1. Two criteria, coefficient of determination (R^2) and reduction degree (RD) are calculated for every selection, as shown in Figure 2.

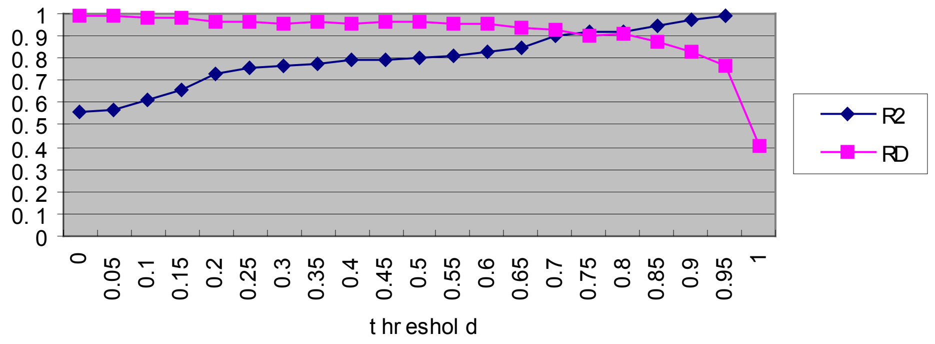


Figure 2: the coefficient of determinant (R^2) and reduction degree (RD) of our strategy data as a function of threshold value using Cactus data.

As expected, when the threshold increases, fewer system metrics group into clusters and thus are removed as redundant. Thus, the reduction degree decreases. At the same time, the coefficient of determination increases since more information is available to model the application performance. However, when the threshold value reaches 1, dependent system metrics are left as potential predictors in the multiple regression model, and we obtain confusing and unreasonable results. The regression failed and no unrelated system metrics was reduced. This problem is called multicollinearity [20]. The reduction degree decrease dramatically to as low as 40% and there is no R^2 value calculated. Thus, before we begin the BE stepwise to delete unrelated predictors, we must identify and reduce the predictors that dependent each other first. Eliminating the redundant predictors can reduce the multicollinearity problem without loss any useful information.

When the threshold value is equal to 0.95, our data reduction strategy produced the highest R^2 value as well as an efficient data reduction rate. On the training data, it produced a reduction degree equal to 0.78 and R^2 as high as 0.98. In other words, 78% of the 628 metrics has been eliminated and the system metrics that remain can explain 98% of the variation in the application performance metric.

A total of 141 of the original 628 system metrics were selected on the six machines when the threshold value was set to 0.95, i.e., about 24 system metrics per machine on average. Each of these metrics is related linearly to Cactus performance. An example of the system metrics selected on one machine is shown in Table 1. We see that in addition to CPU, network and memory measurements, which are commonly used to model application performance, cache, system page, and signal measurements are also important for modeling Cactus performance.

Table1: System metrics selected for Cactus on cirque.ucsd.edu with a threshold value of 0.95.

Name	Measurement
wtps	Total number of write requests per second issued to the physical disk.
activepg	Number of active (recently touched) pages in memory
proc/s	Total number of processes created per second.
rxpck/s	Total number of packets received per second
txpck/s	Total number of packets transmitted per second.
coll/s	Number of collisions that happened per second while transmitting packets.
kbbuffers	Amount of memory used as buffers by the kernel in kilobytes.
ip-frag	Number of IP fragments currently in use.
runq-sz	Run queue length (number of processes waiting for run time)
ldavg-5	System load average for the past 5 minutes.
ldavg-15	System load average for the past 15 minutes.
campg/s	Number of additional memory pages cached by the system per second.
dentunusd	Number of unused cache entries in the directory cache.
file-sz	Number of used file handles.
Rtsig-sz	Number of queued RT signals.
cswch/s	Number of context switches per second.
Latency	Amount of time required to transmit a TCP message to target machine
bandwidth	Speed with which data can be sent to a target machine per second
AvailCPU	Fraction of CPU available to a newly-started process.
FreeMem	Amount of space unused in memory

In the second step of our experiment, we validated our strategy by comparing its result to those of other two strategies, MAIN and RAND, using the remaining 11 chunks of data as the verification data. The coefficient of determination (R^2) was calculated for every strategy, as shown in Figure 3.

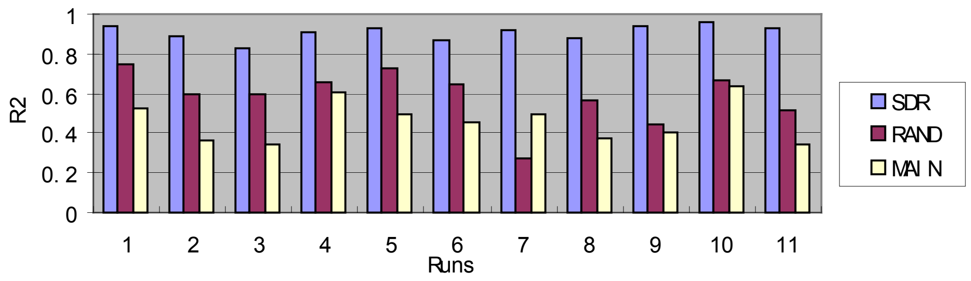


Figure 3: Coefficient of determinant values when regressing Cactus performance on the system metrics selected by three strategies.

We see from Figure 3 that over the 11 chunks of data, our statistical data reduction strategy exhibited an average R^2 value of 0.907, with a range of 0.831 to 0.957. This result is 55.0% and 98.5% higher than those of RAND and MAIN, which have an average R^2 value of 0.585 and 0.457, respectively. We conclude that the system metrics selected by our strategy are significantly more efficient than the alternatives for predicting Cactus performance.

4.4 Sweep3D Results

We ran sweep3D on a shared cluster of six Linux machines at the University of Chicago and again collected data for roughly one day, partitioned into 12 roughly equal-sized chunks each containing about 2 hour's data. Every data point includes values of 689 system metrics and one application performance metric. A detailed description of the system metrics used is available online [23].

We used the first chunk of data as the training data to select the subset of system metrics, while changing the threshold value from 0 to 1 as we did for Cactus. However, the results were not good: the highest R^2 value achieved when the threshold value is equal 0.95 is only 0.79. Furthermore, this subset of the system metrics produced a mean R^2 value equal to 0.72 on the 11 chunks of verification data. The only moderate coefficient of determinant value indicates that we are lacking some information when modeling sweep3D performance. We hypothesized that the problem might be that a linear model was inadequate and thus we extended the linear model to include quadratic items as described in function 2 in Section 3.2.

We redid the experiment using the new model including quadratic items on the sweep3D data. The regression method requires the number of data points used must be larger than the number of potential predictors considered in the model. Because we now have more potential predictors, we may need more data points as training data. We used the first two chunks of data as training data and calculated the coefficient of determinant and reduction degree as a function of threshold value. The results are shown in Figure 4. We now obtain far better results. As with Cactus, the reduction degree decrease and the R^2 value increases as the threshold value increases, except that when the threshold value reaches 1, the related system metric reduction fails due to multicollinearity.

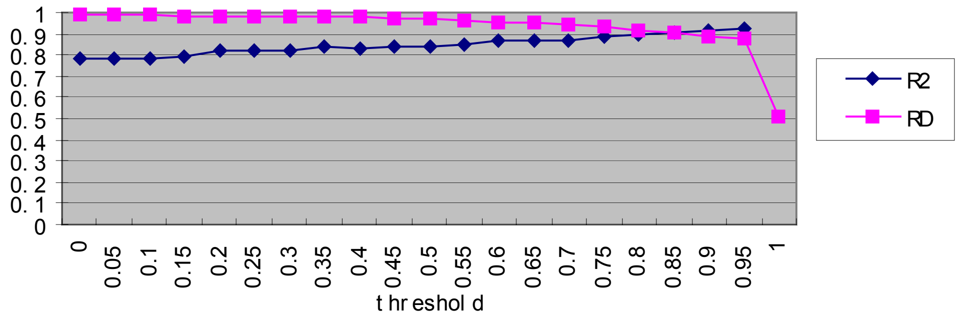


Figure 4: The coefficient of determinant (R^2) and reduction degree (RD) of sweep3D data as a function of threshold.

When the threshold value is equal to 0.95, our data reduction strategy achieved a data reduction degree of 0.88 and a R^2 value as high as 0.928, which indicates that 92.8% of the variation in sweep3D performance is explained by the result metrics selected on linear and/or quadratic relations. 87.8% of the 690 metrics has been reduced by our strategy and only 84 system metrics are left on six machines, 14 on average on each machine. An example of the system metrics selected on one machine is shown in Table 2. We see that in addition to CPU, network, ad memory measurements, the system page and interrupt measurements are also important for sweep3d. In addition, we note that the CPU-load related system metrics (plist-s, %ideal, cswch/s availCPU) are quadratic in the regression model. This result may indicate that CPU capability is quadratically related to sweep3d performance.

Table 2: System metrics selected for sweep3d on broker.cs.uchicago.edu with a threshold value of 0.95.

Name	Measurement	Items included
Tps	Number of transfers per second issued to the physical disk	Linear, quadratic
Activepg	Number of active (recently touched) pages in memory	Linear
intr/s	Report statistics for a given interrupt.	Linear, quadratic
lo	Total number of packets received per second	Linear
Totsck	Total number of used sockets	Quadratic
Plist-s	Number of processes in the process list	Quadratic
Kbcached	Memory used to cache data by the kernel	Linear
%idle	Percentage of time that the CPU(s) were idle and the system did not have an outstanding disk I/O request	Quadratic
cswch/s	Total number of context switches per second	Quadratic
connectTime	Time required to establish a TCP connection to a target sensor	Linear
AvailCPU	Fraction of CPU available to a newly-started process	Quadratic
FreeMem	Amount of space unused in memory	Linear
FreeDisk	Amount of space unused on a disk	Linear, quadratic

We then examine the stability of this result using the left 10 chunks of verification data. Both the linear and quadratic item of every system metric selected is treated as a potential predictor when modeling the performance of the sweep3D application. The coefficient of determination (R^2) of three data reduction strategies were calculated, as shown in Figure 5.

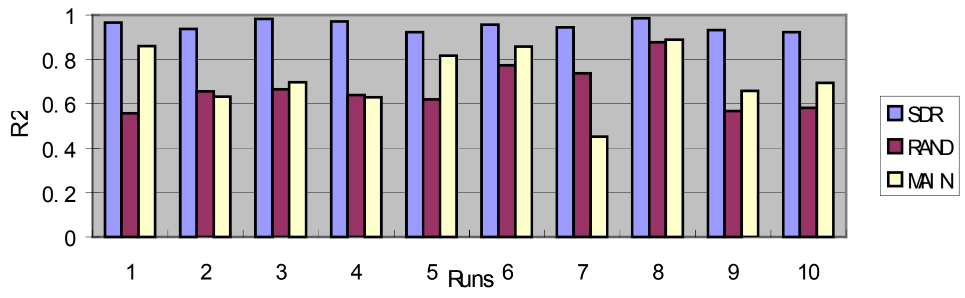


Figure 5: The coefficient of determinant value when regressing the performance of sweep3D application on the system metrics selected by three strategies.

The results in Figure 5 show that the quadratic model improved the performance of our strategy considerably for sweep3D: over 10 chunks of data, our statistical data reduction strategy achieved a mean R^2 value of 0.952 (from 0.918 to 0.985). This result is 42.7% and 32.4% higher than those of the RAND and MAIN strategies, which have average R^2 values of 0.667 and 0.719, respectively.

We also noted that the improvement of our method relative to MAIN strategy is less significant than for Cactus application. This result indicates that MAIN strategy, which includes 75 system metrics that are commonly used to model the performance of applications, can capture the performance of Sweep3D better than Cactus application.

To summarize our results: our stepwise regression-based data reduction strategy achieved better results than the two other strategies considered for two different applications. For both applications examined, the system metrics selected retain enough information to provide useful predictions of application performance, thus dramatically reducing data volume and instrumentation perturbation.

5 Related Work

Data management and reduction have been widely studied in many areas, including medical data analysis [15], financial time series prediction [9], and biological data sampling [24]. Data reduction strategies chiefly rely on statistical techniques such as averaging, variance, covariance matrices, sampling, and principal component analysis (PCA).

In the area of application performance monitoring and analysis, event throttling [16] can replace event tracing with less invasive measures like counts by monitoring the observed event rate and comparing it to user-specified high and low water marks. Although throttling prevents generation of large data volumes, it sacrifices a consistent view of application behavior.

Dynamic clustering [14] identifies clusters of processors with similar performance metric trajectories and then selects one processor from each cluster to represent that cluster and to gather detailed performance information, thus reducing the number of processors or tasks from which even data must be recorded.

Statistical sampling [12] allows the analysis to focus on subsets o[illegible]ocessors or metrics, thus reducing the data to be collected. But its usage is limited to simple cases, such as finding free nodes and estimating average load.

Two approaches that are similar to our data reduction strategy are correlation elimination [7] and projection pursuit [19], both of which identify a relevant statistical interesting subset of system metrics. More specifically, correlation elimination [7] diminishes the volume of performance data by grouping metrics with high correlation coefficient into clusters and only picking one metric for each cluster as a representative. However, that work assumes that all metrics collected are related to the performance metrics. Thus, they only reduce redundant metrics using the correlation elimination technique, as in the first step of our data reduction strategy. In addition, our strategy further improves the correlation elimination technique by using statistic Z-test instead a pure mathematical comparison when trying to group two metrics into one cluster.

Projection pursuit [19] focuses performance analysis on interesting performance metrics. However, projection pursuit selects interesting metrics from all smoothed input data at some discrete point in time, and thus only captures transient relationships between data. The results of projection pursuit vary with time. In addition, the cited work only presented data reduction results using a total of 12 system metrics. In contrast, our strategy tries to capture inherent relationships between data using a stepwise regression-based data reduction method. Our experiments show that the system metrics selected by our strategy are able to capture the variation in the application performance over a far longer time: 24 hours. In addition, our method was tested on a much larger set of system metrics.

Our strategy can be used as a complementary to the dynamic clustering strategy, which reduces the number of processors to be monitored, by providing a means of further reducing the data volume that must be managed on the selected processors.

6 Conclusion

Computer systems and applications are growing more complex. Consequently, performance monitoring and analysis have become more difficult due to the complex interrelationships among runtime components. We present a statistical data reduction strategy that can be used to identify system metrics that are both necessary and sufficient to capture application behavior, thus dramatically reducing the number of system metrics that must be managed.

Our work comprises two steps. First, we show how to reduce redundant system metrics using correlation elimination and Z-test. The result of this step is a set of independent system metrics. Then, we show how to identify system metrics that are related to an application performance metric by a stepwise regression-based technique. We applied our data reduction strategy to data collected from two applications. We find that our strategy reduces about 80% of total system metrics and that the remaining system metrics can explain as high as 90% of the application performance variation 32%~98% more than metrics selected by other strategies.

To date, we have only tried linear and quadratic regression models, but other models can easily be included. We also plan to use this strategy in an anomaly detection system. As next step, we will collect data on a large number of Grid3 sites and use this statistical data reduction strategy as the first step of anomaly detection. Our strategy is able to identify a minimal set of system metrics that capture enough information for modeling application performance. By monitoring and analyzing the system metrics selected, we can diagnose the reason of anomaly application behavior if it happens.

Acknowledgements

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

References

[1] SYSSTAT utiliteis home page http://perso.wanadoo.fr/sebastien.godard/.

[2] Allen, G., Benger, W., Dramlitsch, T., Goodale, T., Hege, H.-C., Lanfermann, G., Merzky, A., Radke, T., Seidel, E. and Shalf, J., Cactus Tools for Grid Applications, *Cluster Computing*, 4 (2001) 179-188.

[3] Allen, G., Benger, W., Goodale, T., Hege, H.-C., Lanfermann, G., Merzky, A., Radke, T., Seidel, E. and Shalf, J., The Cactus Code: A Problem Solving Environment for the Grid. *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC9)*, Pittsburgh, 2000.

[4] Dail, H., Casanova, H. and Berman, F., A Decoupled Scheduling Approach for Grid Application Development Environments. *Supercomputing 2002*, Baltimore, 2002.

[5] Dail, H.J., A Modular Framework for Adaptive Scheduling in Grid Application Development Environments. *Computer Science*, University of California, California, San Diego, 2001.

[6] Foster, I., Gieraltowski, J., Gose, S., Maltsev, N., May, E. and Rodriguez, A., The Grid2004 Production Grid: Principles and Practice. *IEEE International Symposium on High Performance Distributed Computing (HPDC) 2004*, IEEE, Honolulu, Hawaii USA, 2004.

[7] Knop, M.W., Schopf, J.M. and Dinda, P.A., Windows Performance Monitoring and Data Reduction Using WatchTower. *Proceedings of SHAMAN*, 2002.

[8] Koch, K.R., R.S.Baker and Alcouffe, R.E., Solution of the First-order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor, *Trans. Amer. Nuc. Soc.*, 65 (1992).

[9] Lendasse, A., Lee, J., Bodt, E.d., Wertz, V. and Verleyen, M., Input Data Reduction for the Prediction of Financial Time series.

[10] Liu, C., Yang, L., Foster, I. and Angulo, D., Design and Evaluation of a Resource Selection Framework for Grid Applications. *Proceedings of the 11th IEEE International Symposium on High-Performance Distributed Computing (HPDC 11)*, Edinburgh, Scotland, 2002.

[11] Malony, A.D., Reed, D.A. and Wijshoff, H.A.G., Performance Measurement Intrusion and Perturation Analysis, *IEEE Trans. Parallel and Distributed System*, 3 (1992) 433-50.

[12] Mendes, C.L. and Reed, D.A., Monitoring Larger Systems Via Statistical Sampling. *Proceedings of the LACSI Symposium*, Santa Fe, 2002.

[13] Nabrzyski, J., Schopf, J.M. and Weglarz, J., *Grid Resource Management:State of the Art and Future Trends*, Kluwer Publishing, 2003.

[14] Nickolayev, O.Y., Roth, P.C. and Reed, D.A., Real-Time Statistical Clustering For Event Trace Reduction, *The International Journal of Supercomputer Applications and High Performance Computing*, 11 (1997) 144-159.

[15] Qu, Y., Adam, B.-l., Thornquist, M., Potter, J.D., Thompson, M.L., Yasui, Y., Davis, J., Schellhammer, P., Cazares, L., Clements, M., Jr., G.L.W. and Feng, Z., Data Reduction Using a Discrete Wavelet Transform in Discriminant Analysis of Very High Dimensionality Data, *Biometrics*, 59 (2003) 143.

[16] Reed, D.A., Aydt, R.A., Noe, R.J., Roth, P.C., Shields, K.A., Schwartz, B.W. and Travera, L.F., Scalable Performance Analysis: The pablo Performance Analysis Environment. *Proceedings of the Scalable Parallel Libraries Conference*, 1993.

[17] Ripeanu, M., Iamnitchi, A. and Foster, I., Performance Predictions for a Numerical Relativity Package in Grid Environments, *International Journal of High Performance Computing Applications*, 15 (2001).

[18] Stockburger, D.W., *Introductory Statistics: Concepts, Models, and Applications* *http://www.psychstat.smsu.edu/introbook/sbk17.htm*, 1996.

[19] Vetter, J.S. and Reed, D.A., Manging Performance Analysis with Dynamic Statistical Projection Pursuit. *Proceedings of SC'99*, Portland, OR, 1999.

[20] Weisberg, S., *Applied Linear Regression*, 2 edn., John Wiley &Sons, 1985.

[21] Wolski, R., Dynamically Forecasting Network Performance Using the Network Weather Service, *Journal of Cluster Computing* (1998).

[22] Wolski, R., Spring, N. and Hayes, J., The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, *Journal of Future Generation Computing Systems* (1998) 757-768.

[23] Yang, L., Document: http://www.cs.uchicago.edu/~lyang/work/MetricsList.doc, (2004).

[24] Yoccoz, N.G., J.D.Nichols and Boulinier, T., Monitoring of Biological Diversity in space and time, *Trends in Ecology and Evolution*, 16 (2001) 446-453.

[ITF1]What do you mean by “analysis”?